UNITED STATES PATENT AND TRADEMARK OFFICE

UNITED STATES DEPARTMENT OF COMMERCE
United States Patent and Trademark Office
Address: COMMISSIONER FOR PATENTS
P.O. Box 1450
Alexandria, Virginia 22313-1450
www.uspto.gov

APPLICATION NO.	FILING DATE	FIRST NAMED INVENTOR	ATTORNEY DOCKET NO.	CONFIRMATION NO.
10/787,005	02/25/2004	Takeshi Ogasawara	JP920030021US1	3932

48233 7590 10/16/2007
SCULLY, SCOTT, MURPHY & PRESSER, P.C.
400 GARDEN CITY PLAZA
SUITE 300
GARDEN CITY, NY 11530

EXAMINER	
BODDEN, EVRAL E	

ART UNIT	PAPER NUMBER
2192	

MAIL DATE	DELIVERY MODE
10/16/2007	PAPER

Please find below and/or attached an Office communication concerning this application or proceeding.

The time period for reply, if any, is set in the attached communication.

Office Action Summary	Application No.	Applicant(s)	
	10/787,005	OGASAWARA ET AL.	
	Examiner	Art Unit	
	Evral Bodden	2192	

-- The MAILING DATE of this communication appears on the cover sheet with the correspondence address --

Period for Reply

A SHORTENED STATUTORY PERIOD FOR REPLY IS SET TO EXPIRE 3 MONTH(S) OR THIRTY (30) DAYS, WHICHEVER IS LONGER, FROM THE MAILING DATE OF THIS COMMUNICATION.

- Extensions of time may be available under the provisions of 37 CFR 1.136(a). In no event, however, may a reply be timely filed after SIX (6) MONTHS from the mailing date of this communication.
- If NO period for reply is specified above, the maximum statutory period will apply and will expire SIX (6) MONTHS from the mailing date of this communication.
- Failure to reply within the set or extended period for reply will, by statute, cause the application to become ABANDONED (35 U.S.C. § 133). Any reply received by the Office later than three months after the mailing date of this communication, even if timely filed, may reduce any earned patent term adjustment. See 37 CFR 1.704(b).

Status

1)☒ Responsive to communication(s) filed on *07-13-2007*.

2a)☐ This action is **FINAL**. 2b)☒ This action is non-final.

3)☐ Since this application is in condition for allowance except for formal matters, prosecution as to the merits is closed in accordance with the practice under *Ex parte Quayle*, 1935 C.D. 11, 453 O.G. 213.

Disposition of Claims

4)☒ Claim(s) *1 - 24* is/are pending in the application.

4a) Of the above claim(s) *1, 3, 11-18, 21, and 22* is/are withdrawn from consideration.

5)☐ Claim(s) _____ is/are allowed.

6)☒ Claim(s) *2, 4 - 10, 19, 20, 23, and 24* is/are rejected.

7)☐ Claim(s) _____ is/are objected to.

8)☐ Claim(s) _____ are subject to restriction and/or election requirement.

Application Papers

9)☒ The specification is objected to by the Examiner.

10)☐ The drawing(s) filed on _____ is/are: a)☐ accepted or b)☐ objected to by the Examiner.

Applicant may not request that any objection to the drawing(s) be held in abeyance. See 37 CFR 1.85(a).

Replacement drawing sheet(s) including the correction is required if the drawing(s) is objected to. See 37 CFR 1.121(d).

11)☐ The oath or declaration is objected to by the Examiner. Note the attached Office Action or form PTO-152.

Priority under 35 U.S.C. § 119

12)☐ Acknowledgment is made of a claim for foreign priority under 35 U.S.C. § 119(a)-(d) or (f).

a)☐ All b)☐ Some * c)☐ None of:

1.☐ Certified copies of the priority documents have been received.

2.☐ Certified copies of the priority documents have been received in Application No. _____.

3.☐ Copies of the certified copies of the priority documents have been received in this National Stage application from the International Bureau (PCT Rule 17.2(a)).

* See the attached detailed Office action for a list of the certified copies not received.

Attachment(s)

1) ☐ Notice of References Cited (PTO-892)	4) ☐ Interview Summary (PTO-413) Paper No(s)/Mail Date. _____ .
2) ☐ Notice of Draftsperson's Patent Drawing Review (PTO-948)	5) ☐ Notice of Informal Patent Application
3) ☐ Information Disclosure Statement(s) (PTO/SB/08) Paper No(s)/Mail Date _____.	6) ☐ Other: _____.

DETAILED ACTION

1. This action is in response to the following communication: Amendment to application 10/787,005 filed Jul. 13, 2007.

Claims 1, 3, 11 - 18, 21, and **22** have been cancelled.

Claims 2, 4, 5, 9, 19, and **20** have been amended. **Claims 23** and **24** have been added.

Claims 2 – 10, 19, 20, 23, and **24** remain pending.

As an initial matter, examiner notes that in the response filed Jul. 13, 2007, the status of claims in the application under section Listing of Claims (pp. 3 – 8) does not reflect Applicant's statements and/or arguments presented under Remarks section (pp. 9 – 12).

For example, the claims has been indicated as follows:

• Listing of Claims: (pp. 3 – 8)	- Claim 1, 11 – 18, and 21 – 22 have been canceled - Claim 23 and 24 have been added.
• Remarks: (pp. 9 – 12)	- p. 9, 1st para., states "Claims 1 – 22 remain pending. Claims 1, 4, 5, 11, 12, 13, and 19 are independent"! - p. 9, last para., states "... Claim 14 is being amended to delete that word as..."! - p. 10, 3rd para., states "Claims 11 – 18 and 21- 22 being canceled. New claims 23 – 24 are being added..."!

- p. 11, 2nd para. states "Claims 1 and 3 were rejected under 35 USC 102(e)... without conceding ... claims 1 and 3 are being canceled, claim 2 is being rewritten"!

Appropriate status of claims is required.

Specification

2. The abstract of the disclosure was objected to due informalities:

A: Prior objection overcome due to correction.

The disclosure was objected to due to informalities:

A: Prior objection overcome due to correction.

B: Prior objection overcome due to correction.

Claim Objections

A: Prior objection overcome due to correction.

Claim Rejections - 35 USC § 112

3. **Claim 5 and 9** were rejected under 35 U.S.C. 112, first paragraph, as failing to comply with the enablement requirement. Prior objection overcome due to correction.

Claim Rejections - 35 USC § 101

4. **Claims 2, 4, 19,** and **20** were rejected under 35 U.S.C. 101 because

the claimed invention is directed to non-statutory subject matter. Prior rejection is overcome.

Claims 5-10 were rejected under 35 U.S.C. 101 because the claimed invention is directed to non-statutory subject matter. Prior rejection is overcome.

Claims 2, 4, and **5 - 10** were rejected under 35 U.S.C. 101 because the claimed invention is directed to non-statutory subject matter. Prior rejection is overcome.

Claims 2, 4, 5 – 10, 19, and **20** were also rejected under 35 U.S.C. 112, first paragraph. Prior rejection is overcome.

Claim Rejections - 35 USC § 102

5. The following is a quotation of the appropriate paragraphs of 35 U.S.C. 102 that form the basis for the rejections under this section made in this Office action:

> A person shall be entitled to a patent unless –
>
> (b) the invention was patented or described in a printed publication in this or a foreign country or in public use or on sale in this country, more than one year prior to the date of application for patent in the United States.

Claims 2, 3, 4, 5 - 10, 19, 20, 23, and **24** are rejected under 35 U.S.C. 102(b) as being unpatentable over Arnold et al. (hereinafter Arnold) 6,523,168.

In regards to **claim 2,** Arnold teaches:

A compiler device for optimizing a program which manipulates a character string, the compiler device comprising:

a processor including at least an append instruction detection unit for detecting an append instruction to append a character string to a string variable for storing a character string, in the program

Due to the reference to detecting a conflicting operation in Arnold on (column 13, line 22 - 23):

- *detecting a conflicting operation in the first computer*

and in Arnold on (column 12, line 56)

- *wherein each of the plurality of operations includes a string concatenation operation.*

it's inherent that character stings concatenations are detected.

a store code generation unit for generating, as a substitute for each of a plurality of the append instructions detected by the append instruction detection unit, a store code for storing data of an appendant character string to be appended to the string variable by the append instruction into a buffer, the plurality of append instructions appending the character strings to the same string variable; and the processor further including

Due to the reference to generating program code for each of a plurality of operation in Arnold
on (column 12 lines 45 –57):

> *(a) generating program code for the second computer program that allocates a reusable temporary object for handling multiple operations in the second computer program that require the use of temporary storage; and (b) for each of a plurality of operations defined in the first computer program that require the use of temporary storage, generating program code in the second computer program to utilize the reusable temporary object in performing the operation.*

and in Arnold on (column 12, line 56)

- *wherein each of the plurality of operations includes a string concatenation operation.*

, it's inherent store code/program code for the plurality of operations/appended instructions is being generated.

an append code generation unit for generating an append code for appending a plurality of the appendant character strings to the string variable, at a position to be executed before an instruction to refer to the string variable in the program

Due to the reference to reduction of multiple sting concatenation in Arnold on (column 4, lines 19-28):

- *rather than creating a new mutable string object (as well as an underlying character array object) for each string concatenation operation, an existing mutable string object, allocated at the initialization of a program (or a thread thereof), is used as the temporary storage for each operation. The total number of objects created as a result of multiple string concatenation operations is therefore reduced.*

it's inherent that a plurality of appendant character strings are appended.

a reference instruction detection unit for detecting a reference instruction which first refers to the string variable after the character strings have been appended to the string variable by the plurality of append instructions, wherein the append code generation unit generates the append code at a position to be executed after the store codes and before the reference instruction

Due to the reference to tbs.append(s1), append(s2), and toString() in Arnold on (column 10, lines 35 – 67):

- *Returning to blocks 92 and 94, if no argument to the string concatenation operation includes a method call, or if no method called by an argument includes a string concatenation operation, control passes to block 98 to generate temporary StringBuffer usage code to utilize the reusable temporary StringBuffer object in the performance of the string concatenation operation, prior to terminating routine 84.*

  A source code representation of one suitable temporary StringBuffer usage code implementation is illustrated in Table IV below:

  TABLE IV
  Temporary StringBuffer Usage Code
  ```
  tsb.setLength(0);
  String result = tsb.append(s1).append(s2).toString();
  ```

  The first statement "tsb.setLength(0)" simply clears the reusable temporary StringBuffer object for the thread. The second statement "String result=tsb.append(s1).append(s2).toString( )" can be parsed into the following operations: 1. tsb.append(s1)--appends "s1" to the reusable temporary StringBuffer object 2. .append(s2)--appends "s2" to the reusable temporary StringBuffer object 3. .toString( )--creates a new "String" object from the reusable temporary StringBuffer object.

  It will be appreciated that if more than two arguments are supplied to a string concatenation operations, the program code of Table IV would simply require the use of additional "appends" operations to concatenate the additional arguments.

, in addition to Fig. 4, # 74, 78, and 84, it's inherent that append instructions are being detected.

In regards to **claim 3**, Arnold teaches a:

- The compiler device according to claim 1, wherein the append instruction detection unit detects, as the append instruction, a combination of: an instruction to convert an immutable string variable in which a process of appending a character string is not allowed, into a mutable string variable in which a process of appending a character string is allowed; an instruction to append the appendant character string to the mutable string variable; and an instruction to convert the mutable string variable into the immutable string variable.

Due to the reference to detecting a conflicting operation in Arnold on (column 13, line 22):

- *detecting a conflicting operation in the first computer*

, and the reference to plurality of string concatenation operation in Arnold on (column 12, lines 45-57):

- *translating a first computer program into a second computer program, the method comprising: (a) generating program code for the second computer program that allocates a reusable temporary object for handling multiple operations in the second computer program that require the use of temporary storage; and (b) for each of a plurality of operations defined in the first computer program that require the use of temporary storage, generating program code in the second computer program to utilize the reusable temporary object in performing the operation.*
- *wherein each of the plurality of operations includes a string concatenation operation.*

and the reference to includes mutable string object in Arnold on (column 14, lines 45-46) :

- *wherein the reusable temporary object includes a mutable string object.*

, it's inherrent that append instructions applicable to mutable character strings are being detected, and optimized.

In regards to **claim 4,** Arnold teaches:

A compiler device for optimizing a program which manipulates a character string

The reference to compiler in (column 5 lines 37 – 38):

- *A translation program consistent with the invention may incorporate a compiler to generate suitable program code.*

and the reference to string concatenation in (column 7, 63-64):

- *Reduction of Object Creation During String Concatenation*

implies that the optimization of a compiler is achieved by targeting character strings.

append instruction detection unit for detecting an append instruction to append a character string to a string variable for storing a character string, in the program

The reference to detecting a conflicting operation in Arnold on (column 13, line 22):

- *detecting a conflicting operation in the first computer*

and in Arnold on (column 12, line 56)

- *wherein each of the plurality of operations includes a string concatenation operation.*

implies that character stings concatenations are detected.

a store code generation unit for generating, as a substitute for each of a plurality of the append instructions detected by the append instruction detection unit, a store code for storing an address in memory where an appendant character string to be appended to the string variable by the append instruction is stored, into a buffer, the plurality of append instructions appending character strings to the same string variable; and the processor further including.

The reference to generating program code for each of a plurality of operation in Arnold on (column 12 lines 45 –57):

> *(a) generating program code for the second computer program that allocates a reusable temporary object for handling multiple operations in the second computer program that require the use of temporary storage; and (b) for each of a plurality of operations defined in the first computer program that require the use of temporary storage, generating program code in the second computer program to utilize the reusable temporary object in performing the operation.*

and in Arnold on (column 12, line 56)

- *wherein each of the plurality of operations includes a string concatenation operation.*

implies that store code/program code for the plurality of operations/appended instructions is being generated.

an append code generation unit for generating an append code for appending a plurality of the appendant character strings stored in a plurality of the addresses, to the string variable, at a position to be executed before an instruction to refer to the string variable in the program.

The reference to reduction of multiple sting concatenation in Arnold on (column 4, lines 19-28):

- *rather than creating a new mutable string object (as well as an underlying character array object) for each string concatenation operation, an existing mutable string object, allocated at the initialization of a program (or a thread thereof), is used as the temporary storage for each operation. The total number of objects created as a result of multiple string concatenation operations is therefore reduced.*

implies that a plurality of appendant character strings are stored.

In regards to **claim 5,** Arnold teaches:

A compiler device for optimizing a program which manipulates a character string,

the compiler device comprising: a processor including at least_a mutable-to-immutable conversion instruction detection unit for detecting a mutable-to-immutable conversion instruction to convert a mutable string variable in which a process of appending a character string is allowed, into an immutable string variable in which a process of appending a character string is not allowed; the processor further including
an immutable-to-mutable conversion instruction detection unit for detecting an immutable-to-mutable conversion instruction to convert the immutable string variable into the mutable string variable; and the processor further including
an instruction elimination unit for eliminating the immutable-to-mutable conversion instruction and for causing the mutable string variable to be used as the mutable string variable obtained after the immutable-to-mutable conversion instruction, if an instruction to be executed between the mutable-to-immutable conversion instruction and the immutable-to-mutable conversion instruction does not modify a character string stored in the mutable string variable, and if an instruction to be executed between the immutable-to-mutable conversion instruction and use of the mutable string variable instruction does not modify any of the mutable string variable used as the source variable of the mutable-to- immutable conversion instruction and the mutable string variable

The reference to detecting a conflicting operation in Arnold on (column 13, line 22):

- *detecting a conflicting operation in the first computer*

the reduction of multiple sting concatenation in Arnold on (column 4, lines 19-28):

- *rather than creating a new mutable string object (as well as an underlying character array object) for each string concatenation operation, an existing mutable string object, allocated at the initialization of a program (or a thread thereof), is used as the temporary storage for each operation. The total number of objects created as a result of multiple string concatenation operations is therefore reduced, easing allocation and collection overhead, and accordingly improving overall system performance.*

and in Arnold on (column 12 lines 45 –57):

- *translating a first computer program into a second computer program, the method comprising: (a) generating program code for the second computer program that allocates a reusable temporary object for handling multiple operations in the second computer program that require the use of temporary storage; and (b) for each of a plurality of operations defined in the first computer program that require the use of temporary storage, generating program code in the second computer program to utilize the reusable temporary object in performing the operation.*

implies that character strings concatenations are detected, stored, and optimized via the reduction of multiple sting concatenations.

In addition, the reference to the translation process being utilized using any number of representations in Arnold on (column 5, line 65-67):

- *being translated, as well as the computer program being generated as a result of the translation process, may be utilized using any number of representations, whether human or machine readable in nature.*

implies that various methods are utilized to detect, append, and optimize various types of character strings.

In regards to **claim 6**, Arnold teaches:

The compiler device according to claim 5, wherein the instruction elimination unit further eliminates the mutable-to-immutable conversion instruction if a character string stored in the immutable string variable is not referred to.

The reference to deallocating in Arnold on (column 1, Lines 26- 31)

- *some mechanism for removing, or "deallocating", unused objects is also provided, typically either through the use of specific program instructions or through an automated process known as garbage collection.*

implies that strings not referenced are deallocated and eliminated.

In regards to **claim 7**, Arnold teaches:

the instruction elimination unit moves the mutable-to-immutable conversion instruction to each branch destination of a branch instruction to be executed after the mutable-to-immutable conversion instruction, and executes partial dead assignment elimination for eliminating the mutable-to- immutable conversion instruction if a character string stored in the immutable string variable as a destination variable of the mutable-to-immutable conversion instruction is not referred to on each branch destination of the branch instruction.

Due to the reference to tbs.append(s1), append(s2), and toString() in Arnold on (column 10, lines 35 – 67):

- *Returning to blocks 92 and 94, if no argument to the string concatenation operation includes a method call, or if no method called by an argument includes a string concatenation operation, control passes to block 98 to generate temporary StringBuffer usage code to utilize the reusable temporary StringBuffer object in the performance of the string concatenation operation, prior to terminating routine 84.*

  A source code representation of one suitable temporary StringBuffer usage code implementation is illustrated in Table IV below:

  TABLE IV
  Temporary StringBuffer Usage Code
  tsb.setLength(0);
  String result = tsb.append(s1).append(s2).toString();

> *The first statement "tsb.setLength(0)" simply clears the reusable temporary StringBuffer object for the thread. The second statement "String result=tsb.append(s1).append(s2).toString( )" can be parsed into the following operations: 1. tsb.append(s1)--appends "s1" to the reusable temporary StringBuffer object 2. .append(s2)--appends "s2" to the reusable temporary StringBuffer object 3. .toString( )--creates a new "String" object from the reusable temporary StringBuffer object.*
>
> *It will be appreciated that if more than two arguments are supplied to a string concatenation operations, the program code of Table IV would simply require the use of additional "appends" operations to concatenate the additional arguments.*

, in addition to Fig. 4, # 74, 78, and 84, it's inherent that append instructions are being detected and optimized.

In regards to **claim 8**, Arnold teaches:

The compiler device according to claim 5, wherein the immutable-to-mutable conversion instruction detection unit detects, as the immutable-to-mutable conversion instruction, a combination of: an instruction to reserve a memory area to be used as a mutable string variable; and an instruction to append a character string stored in the immutable string variable to the mutable string variable.

the reference to detecting a conflicting operation in Arnold on (column 13, line 22):

- *detecting a conflicting operation in the first computer*

and the reference to allocating memory for string concatenation operation in Arnold in (column 14, line 15-23):

- *the use of temporary storage, to generate program code in the second computer program to utilize the reusable temporary object in performing the operation.*
- *The apparatus of claim 17, wherein each of the plurality of operations includes a string concatenation operation.*

implies that memory is being reserved for the purpose of optimizing character strings.

In regards to **claim 9,** Arnold teaches:

The compiler device according to claim 5, wherein the processor further comprises comprising: a partial redundancy elimination unit for executing a partial redundancy elimination process of moving the immutable-to-mutable conversion instruction detected by the immutable-to-mutable conversion instruction detection unit to each control flow edge which merges into a single control flow before the immutable-to-mutable conversion instruction, wherein, in a program obtained after the partial redundancy elimination process has been executed, the instruction elimination unit eliminates the immutable-to-mutable conversion instruction, if an instruction to be executed between the mutable-to-immutable conversion instruction and the immutable-to-mutable conversion instruction does not modify a character string stored in the mutable string variable used as the source variable of the mutable-to-immutable conversion instruction, and if an instruction to be executed between the immutable-to-mutable conversion instruction and the use of the mutable string variable obtained from the conversion by the immutable-to-mutable conversion instruction does not modify any of the mutable string variable being used as the source variable of the mutable-to-immutable conversion instruction and the mutable string variable obtained from the conversion by the immutable-to-mutable conversion instruction.

, the reference to detecting a conflicting operation in Arnold on (column 13, line 22):

- *detecting a conflicting operation in the first computer*

, the reference to plurality of string concatenation operation in Arnold on (column 12, lines 45-57):

- *translating a first computer program into a second computer program, the method comprising: (a) generating program code for the second computer program that allocates a reusable temporary object for handling multiple operations in the second computer program that require the use of temporary storage; and (b) for each of a plurality of operations defined in the first computer program that require the use of temporary storage, generating program code in the second computer program to utilize the reusable temporary object in performing the operation.*

and the reference to includes mutable string object in Arnold on (column 14, lines 45-46):

- *wherein each of the plurality of operations includes a string concatenation operation.*

implies that append instructions applicable to mutable character strings are being detected, and optimized.

In regards to **claim 10**, Arnold teaches:

The instruction elimination unit moves the mutable-to-immutable conversion instruction to each branch destination of a branch instruction to be executed after the mutable-to-immutable conversion instruction, and executes partial dead assignment elimination for eliminating the mutable-to- immutable conversion instruction if a character string stored in the immutable string variable as a destination variable of the mutable-to-immutable conversion instruction is not referred to on each branch destination of the branch instruction.

Due to the reference to tbs.append(s1), append(s2), and toString() in Arnold on (column 10, lines 35 – 67):

- *Returning to blocks 92 and 94, if no argument to the string concatenation operation includes a method call, or if no method called by an argument includes a string concatenation operation, control passes to block 98 to generate temporary StringBuffer usage code to utilize the reusable temporary StringBuffer object in the performance of the string concatenation operation, prior to terminating routine 84.*

  A source code representation of one suitable temporary StringBuffer usage code implementation is illustrated in Table IV below:

  TABLE IV
  Temporary StringBuffer Usage Code
  tsb.setLength(0);
  String result = tsb.append(s1).append(s2).toString();

  The first statement "tsb.setLength(0)" simply clears the reusable temporary StringBuffer object for the thread. The second statement "String result=tsb.append(s1).append(s2).toString( )" can be parsed into the following operations: 1. tsb.append(s1)--appends "s1" to the reusable temporary StringBuffer object 2. .append(s2)--appends "s2" to the reusable temporary StringBuffer object 3. .toString( )--creates a new "String" object from the reusable temporary StringBuffer object.

  It will be appreciated that if more than two arguments are supplied to a string concatenation operations, the program code of Table IV would simply require the use of additional "appends" operations to concatenate the additional arguments.

, in addition to Fig. 4, # 74, 78, and 84, it's inherent that append instructions are being detected, eliminated, and optimized.

In regards to **claim 19**, Arnold teaches:

A computer-implemented method for optimizing a program which manipulates a character string

The reference to compiler in (column 5 lines 37 – 38):

- *A translation program consistent with the invention may incorporate a compiler to generate suitable program code.*

and the reference to string concatenation in (column 7, 63-64):

- *Reduction of Object Creation During String Concatenation*

, implies that the optimization of a compiler is achieved by targeting character strings.

the method comprising detecting an append instruction to append a character string to a string variable for storing a character string, in the program;

The reference to detecting a conflicting operation in Arnold on (column 13, line 22):

- *detecting a conflicting operation in the first computer*

and in Arnold on (column 12, line 56)

- *wherein each of the plurality of operations includes a string concatenation operation.*

implies that character stings concatenations are detected.

generating, as a substitute for each of a plurality of the append instructions detected by the append instruction detection unit, a store code for storing data of an appendant character string to be appended to the string variable by the append instruction into a buffer, the plurality of append instructions appending the character strings to the same string variable

The reference to generating program code for each of a plurality of operation in Arnold on (column 12 lines 45 –57):

(a) generating program code for the second computer program that allocates a reusable temporary object for handling multiple operations in the second computer program that require the use of temporary storage; and (b) for each of a plurality of operations defined in the first computer program that require the use of temporary storage, generating program code in the second computer program to utilize the reusable temporary object in performing the operation.

, and in Arnold on (column 12, line 56)

- *wherein each of the plurality of operations includes a string concatenation operation.*

, implies that store code/program code for the plurality of operations/appended instructions is being generated.

generating an append code for appending a plurality of the appendant character strings to the string variable, at a position to be executed before an instruction to refer to the string variable in the program, the append code being an optimized set &instructions for a processor to execute.

The reference to reduction of multiple sting concatenation in Arnold on (column 4, lines 19-28):

- *rather than creating a new mutable string object (as well as an underlying character array object) for each string concatenation operation, an existing mutable string object, allocated at the initialization of a program (or a thread thereof), is used as the temporary storage for each operation. The total number of objects created as a result of multiple string concatenation operations is therefore reduced, easing allocation and collection overhead, and accordingly improving overall system performance.*

implies that a plurality of appendant character strings are stored.

In regards to **claim 20**, Arnold teaches:

Detecting a reference instruction which first refers to the string variable after the character strings have been appended to the string variable by the plurality of append instructions, wherein the append code generation unit generates the append code at a position to be executed after the store codes and before the reference instruction.

Due to the reference to tbs.append(s1), append(s2), and toString() in Arnold on (column 10, lines 35 – 67):

- *Returning to blocks 92 and 94, if no argument to the string concatenation operation includes a method call, or if no method called by an argument includes a string concatenation operation, control passes to block 98 to generate temporary StringBuffer usage code to utilize the reusable temporary StringBuffer object in the performance of the string concatenation operation, prior to terminating routine 84.*

  A source code representation of one suitable temporary StringBuffer usage code implementation is illustrated in Table IV below:

  TABLE IV
  Temporary StringBuffer Usage Code
  tsb.setLength(0);
  String result = tsb.append(s1).append(s2).toString();

  The first statement "tsb.setLength(0)" simply clears the reusable temporary StringBuffer object for the thread. The second statement "String result=tsb.append(s1).append(s2).toString( )" can be parsed into the following operations: 1. tsb.append(s1)--appends "s1" to the reusable temporary StringBuffer object 2. .append(s2)--appends "s2" to the reusable temporary StringBuffer object 3. .toString( )--creates a new "String" object from the reusable temporary StringBuffer object.

  It will be appreciated that if more than two arguments are supplied to a string concatenation operations, the program code of Table IV would simply require the use of additional "appends" operations to concatenate the additional arguments.

, in addition to Fig. 4, # 74, 78, and 84, it's inherent that append instructions are being detected.

In regards to **claim 23**, Arnold teaches:

A program storage device readable by machine, tangibly embodying a program of instructions executable by the machine to perform method steps for optimizing a program which manipulates a character string:

Due to the reference to compiler in (column 5, lines 37 – 38):

- *A translation program consistent with the invention may incorporate a compiler to generate suitable program code.*

and the reference to string concatenation in (column 7, 63-64):

- *Reduction of Object Creation During String Concatenation*

, it's inherent that the optimization of a compiler is achieved by targeting strings.

detecting an append instruction to append a character string to a string variable for storing a character string, in the program; generating, as a substitute for each of a plurality of the append instructions detected by the append instruction detection unit,

Due to the reference to detecting a conflicting operation in Arnold on (column 13, line 22):

- *detecting a conflicting operation in the first computer*

and in Arnold on (column 12, line 56)

- *wherein each of the plurality of operations includes a string concatenation operation.*

it's inherent that character stings concatenations are detected.

a store code for storing data of an appendant character string to be appended to the string variable by the append instruction into a buffer, the plurality of append instructions appending the character strings to the same string variable

Due to the reference to generating program code for each of a plurality of operation in Arnold on (column 12 lines 45 –57):

> *(a) generating program code for the second computer program that allocates a reusable temporary object for handling multiple operations in the second computer program that require the use of temporary storage; and (b) for each of a plurality of operations defined in the first computer program that require the use of temporary storage, generating program code in the second*

computer program to utilize the reusable temporary object in performing the operation.

and in Arnold on (column 12, line 56)

- *wherein each of the plurality of operations includes a string concatenation operation.*

it's inherent store code/program code for the plurality of operations/appended instructions is being generated.

an append code for appending a plurality of the appendant character strings to the string variable, at a position to be executed before an instruction to refer to the string variable in the program, the append code being an optimized set of instructions for a processor to execute.

Due to the reference to reduction of multiple sting concatenation in Arnold on (column 4, lines 19-28):

- *rather than creating a new mutable string object (as well as an underlying character array object) for each string concatenation operation, an existing mutable string object, allocated at the initialization of a program (or a thread thereof), is used as the temporary storage for each operation. The total number of objects created as a result of multiple string concatenation operations is therefore reduced.*

it's inherent that a plurality of appendant character strings are appended.

In regards to **claim 24,** Arnold teaches:

Detecting a reference instruction which first refers to the string variable after the character strings have been appended to the string variable by the plurality of append instructions, wherein the append code generation unit generates the append code at a position to be executed after the store codes and before the reference instruction.

Due to the reference to tbs.append(s1), append(s2), and toString() in Arnold on (column 10, lines 35 – 67):

- *Returning to blocks 92 and 94, if no argument to the string concatenation operation includes a method call, or if no method called by an argument includes a string concatenation operation, control passes to block 98 to generate temporary StringBuffer usage code to utilize the reusable temporary StringBuffer object in the performance of the string concatenation operation, prior to terminating routine 84.*

  A source code representation of one suitable temporary StringBuffer usage code implementation is illustrated in Table IV below:

  TABLE IV
  Temporary StringBuffer Usage Code
  tsb.setLength(0);
  String result = tsb.append(s1).append(s2).toString();

  The first statement "tsb.setLength(0)" simply clears the reusable temporary StringBuffer object for the thread. The second statement "String result=tsb.append(s1).append(s2).toString( )" can be parsed into the following operations: 1. tsb.append(s1)--appends "s1" to the reusable temporary StringBuffer object 2. .append(s2)--appends "s2" to the reusable temporary StringBuffer object 3. .toString( )--creates a new "String" object from the reusable temporary StringBuffer object.

  It will be appreciated that if more than two arguments are supplied to a string concatenation operations, the program code of Table IV would simply require the use of additional "appends" operations to concatenate the additional arguments.

, in addition to Fig. 4, # 74, 78, and 84, it's inherent that append instructions are being detected.

Response to Arguments

Regarding claim 2, after further review of previously applied art, specifically Arnold (column 10, lines 1 - 5), which discloses that "One suitable implementation of routine 84 is illustrated in Fig. 5, where control begins in block 90 by determining the arguments to the string concatenation statement, an operation that is well known in the art", and (column 10, lines 35 – 67, see tbs.append(s1), append(s2), toString(), Fig. 4 # 74, 78, 84, and Fig. 5, 90, 92, and 96), thus a closer reading of Arnold has been applied herein.

Correspondence Information

Any inquiry concerning this communication or earlier communications from the examiner should be directed to Evral Bodden whose telephone number is 571 272 3455. The examiner can normally be reached on Monday to Friday, 8:30 to 5:00.

If attempts to reach the examiner by telephone are unsuccessful, the examiner's supervisor, Tuan Dam can be reached on 571-272-3695. The fax phone number for the organization where this application or proceeding is assigned is 571-273-8300.

Information regarding the status of an application may be obtained from the Patent Application Information Retrieval (PAIR) system. Status information for published applications may be obtained from either Private PAIR or Public PAIR. Status information for unpublished applications is available through Private PAIR only. For more information about the PAIR system, see http://pair-direct.uspto.gov. Should you have questions on access to the Private PAIR system, contact the Electronic Business Center (EBC) at 866-217-9197 (toll-free). If you would like assistance from a USPTO Customer Service Representative or access to the automated information system, call 800-786-9199 (IN USA OR CANADA) or 571-272-1000.

[illegible]

TUAN DAM
SUPERVISORY PATENT EXAMINER

Evral Bodden E. B.